Universitatea Transilvania din Brașov

Universitatea Transilvania din Brașov
FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR

Departamentul: DEC
Programul de studii: SECI-SI

Radu Mihai-Alexandru
Boziesan Stefan
Mitu Mariana Luciana
Ferencz Carnu
Acatrinei Sergiu

PROIECT DE VERIFICARE

Brașov, 2026

Departamentul: DEC
Programul de studii: SECI - SI

Radu Mihai-Alexandru
Boziesan Stefan
Mitu Mariana Luciana
Ferencz Carnu
Acatrinei Sergiu

Controler de intreruperi

Brașov, 2026

FIȘA PROIECTULUI DE VERIFICARE

Universitatea Transilvania din Braşov	Anul universitar:
Facultatea de Inginerie Electrică şi Ştiința Calculatoarelor	2025 - 2026

Membrii: **Numele, prenumele și funcția (arhitect, inginer verificare, proiectant) membrului 1** **Numele, prenumele și funcția membrului 2** **Numele, prenumele și funcția membrului 3** **Numele, prenumele și funcția membrului 4 (opțional)** Coordonatori: **Florin Deneș/Oana Ursu/ Andreea Păscociu/Liviu Manole / Alexandru Dinu (lasati doar numele coordonatorului vostru)**	[poză sugestivă pentru tematica proiectului]  Sursa imaginii: https://www.loc.gov/item/2016632577/
Titlul proiectului*: .....*	
Sumarul funcționalităților DUT-ului: 1. 2. 3. 4.	
Sumarul scenariilor de verificare: 1. 2. 3.	

Istoricul versiunilor (se va completa pe tot parcursul semestrului, atunci când se va modifica specificația:

Ver-siune	Data finalizării	Numele editorului	Motiv
0.0	20.03.2025	Alexandru Dinu	Prima versiune de model
0.1	28.03.2025	Alexandru Dinu	Template-ul a fost actualizat
1.0			Versiunea inițială
1.1			[Exemplu] s-a modificat marimea semnalului ready, de la 2 biți la 1 bit
1.2			[Exemplu] s-a adăugat o nouă stare în automatul de stări care modelează funcționarea DUT-ului în modul “automat”
2.0			O modificare majoră (se spune care) a avut loc

Cuprins

Locația proiectului

Textul aflat în cadrul subcapitolelor este util pentru ghidarea scrierii documentației. Acest text însă nu va trebui să apară în versiunea finală a documentației. De asemenea, și bucățile de text scrise cu roșu mai sus vor trebui șterse. Pentru a exemplifica cum se completează acest template, s-a luat ca exemplu proiectarea unui calculator de numere complexe.

Se vor introduce linkul către mediul din Edaplayground sau pagina de GitHub folosita pentru codul Verilog/SystemVerilog (care trebuie să fie cu acces public), linkul catre mediul din Edaplayground unde s-a dezvoltat DUT-ul (dacă este cazul), datele de conectare la contul Edaplayground folosit de către întreaga echipă și linkul către documentația aflata pe Google Drive.

1 PROIECTUL DIGITAL AL CIRCUITULUI

Diagrama bloc

Descrierea funcționalității generale a DUT-ului

Interfețele DUT-ului

Lista cu funcționalitățile DUT-ului

Forme de undă

Tabel cu regiștri

1.1 DIAGRAMA BLOC

Se realizează un desen cu interfețele DUT-ului. Nu este obligatoriu, în cazul fiecărei interfețe, numele tuturor semnalelor. Daca acestea sunt totusi vizibile in figura, se va marca direcția semnalelor de pe interfață.

Exemple de diagrame:

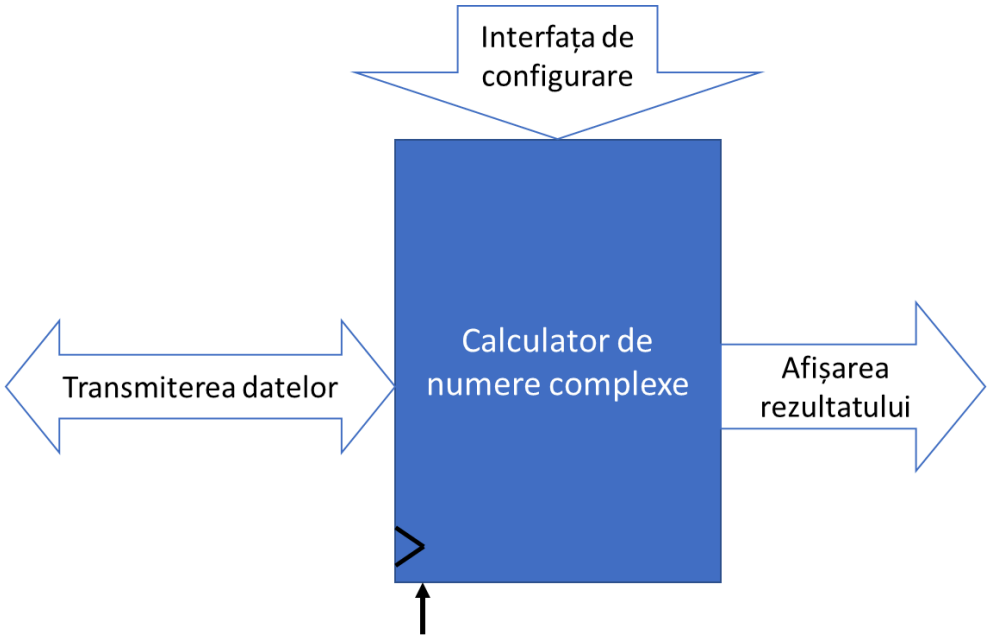


Figura 1 Exemplu de desen cu interfețele unui DUT cu funcția de calculator cu numere complexe. În acest caz interfața pentru transmiterea datelor este bidirecțională deoarece DUT-ul transmite pe interfață semnalul acknowledge

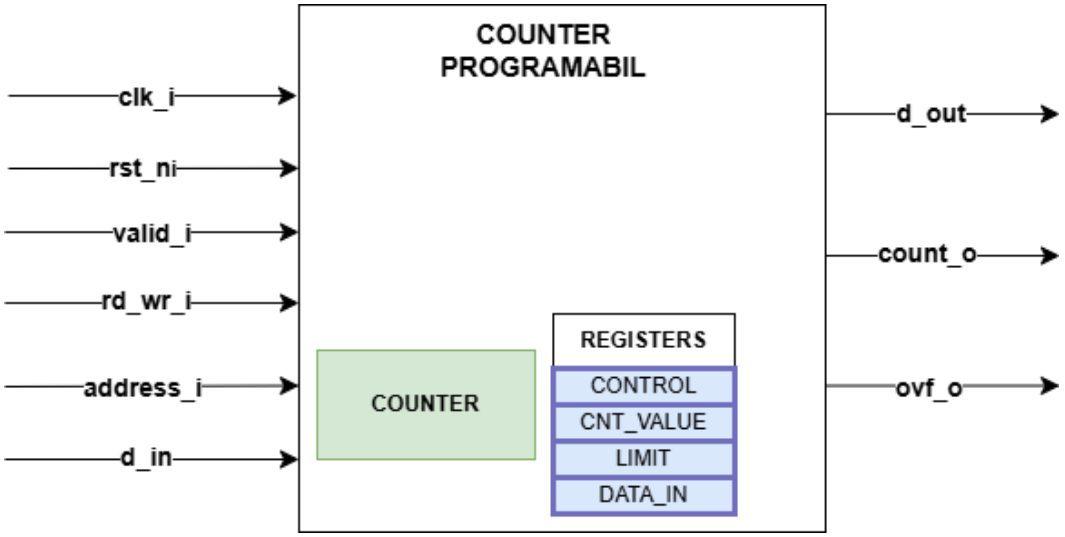


Figura 2. Modulul numărător programabil, care include un contor intern, un set de registre configurabile (CONTROL, CNT_VALUE, LIMIT, DATA_IN) și o interfață de acces pentru citire și scriere. Modulul primește semnalele de ceas și reset, precum și semnale de control pentru operațiile de citire/scriere, și furnizează la ieșire valoarea curentă a contorului (count_o), datele citite din registre (d_out) și semnalul de overflow/underflow (ovf_o).

1.2 Descrierea funcționalității generale a DUT-ului

Modulul Packet Scheduler este un bloc hardware proiectat pentru a colecta, arbitra și ordona cererile (pachetele de date) provenite din partea a 4 clienți distincți (Client 0 ... 3), punându-le la dispoziția unui procesor sau sistem gazdă prin intermediul unei interfețe standard APB.

Interfața cu clienții: Transferul datelor de la clienți către modul se realizează prin intermediul unui bloc Handler, pe baza unui protocol de tip Request/Acknowledge (REQ/ACK).

Stocarea temporară: Datele aferente fiecărei cereri primite sunt stocate imediat în patru cozi FIFO individuale, dedicate fiecărui port de client în parte (Queue 0 ... 3).

Mecanismul de arbitrare: Un bloc logic intern evaluează constant starea acestor cozi de intrare și extrage datele, multiplexându-le într-o coadă unică, centralizată (Output FIFO). Ordinea de preluare a pachetelor de la clienți este dictată de un algoritm de planificare configurabil. Modulul suportă trei moduri de arbitrare:

Strict Priority (SP): Prioritate fixă acordată clienților.

Round Robin (RR): Distribuție circulară și egală.

Weighted Round Robin (WRR): Distribuție bazată pe ponderi programabile.

Control și monitorizare (Interfața APB): Sistemul gazdă poate citi pachetele consolidate direct din Output FIFO folosind magistrala APB. Tot prin APB, utilizatorul configurează algoritmul de arbitrare, setează ponderile pentru WRR, și poate activa sau dezactiva independent fiecare client. De asemenea, modulul raportează în timp real starea sistemului (indicatori de coadă plină/goală) și eventualele erori de transfer (NACK) prin registrele de stare.

1.3 Interfețele DUT-ului

Tabelul 1. Semnalele din cadrul interfeței APB

Nume semnal	Număr de biți	Direcție	Descriere
pclk_i	1	Intrare	Semnalul de ceas utilizat pentru sincronizarea interfeței APB. Toate tranzițiile au loc pe frontul crescător al acestui ceas.
preset_ni	1	Intrare	Semnal de reset asincron, activ în 0 (low), folosit pentru a readuce logica interfeței și registrele APB într-o stare inițială cunoscută.
paddr_i	2	Intrare	Magistrala de adrese. Este folosită de sistemul gazdă pentru a selecta registrul specific la care se dorește accesul.
psel_i	1	Intrare	Semnalul de selecție (Slave Select). Indică faptul că modulul curent este ținta transferului APB de pe magistrală.
penable_i	1	Intrare	Semnalul de activare (Enable). Indică a doua fază a transferului APB și este folosit pentru a valida tranzacția de date.
pwrite_i	1	Intrare	Semnalul de control al direcției. Un nivel logic 1 indică o operație de scriere , iar 0 indică o citire.
pwdata_i	8	Intrare	Magistrala de date pentru scriere. Conține datele care trebuie scrise în registrele interne (ex: ponderile pentru WRR sau bitul de activare client).

prdata_o	*8*	*Ieșire*	*Magistrala de date pentru citire. Oferă datele citite din registrele de stare sau pachetele agregate extrase din Output FIFO.*
pready_o	*1*	*Ieșire*	*Semnal de gata (Ready). Indică finalizarea transferului APB. Dacă Output FIFO este gol la o cerere de citire, acest semnal poate fi ținut în 0 pentru a introduce wait states.*
pslverr_o	*1*	*Ieșire*	*Semnal de eroare (Slave Error). Este activat pe durata pready_o = 1 pentru a indica o eroare (ex: încercarea de scriere într-un registru Read-Only sau accesarea unei adrese nealocate).*

Tabelul 2 Semnalele de pe interfața de Client

Nume semnal	**Număr de biți**	**Direcție**	**Descriere**
req_x_i (x=0..3)	1	Intrare	Semnal de cerere (Request) provenit de la clientul *x*. Indică faptul că datele prezentate pe magistrala data_x_i sunt valide și clientul dorește să le transmită.
data_x_i	*8*	Intrare	Magistrala de date pentru clientul *x*. Conține pachetul propriu-zis de informații.
ack_x_o	1	Ieșire	Semnal de confirmare (Acknowledge). Este activat de modul pentru a informa clientul *x* că

			datele au fost preluate cu succes și stocate în coada sa FIFO.

1.4 Regiștrii modulului – varianta 1

Modulul utilizează un spațiu de adresare de 2 biți, oferind acces la 4 regiștri de câte 8 biți fiecare.

Tabelul 3. Regiștrii DUT-ului

Adresă (Hex)	Nume Registru	Acces	Valoare Reset	Descriere
0x0	**DATA_OUT**	RO	0x00	Portul de citire pachete din Output FIFO.
0x1	**STATUS**	RO	0x04	Starea cozilor.
0x2	**CFG**	RW	0x0F	Algoritm selectat și activare clienți.
0x3	**WEIGHT**	RW	0xE4	Ponderi pentru algoritmul WRR.

Structura registrului de control (**DATA_OUT**) se găsește în Figura 4.

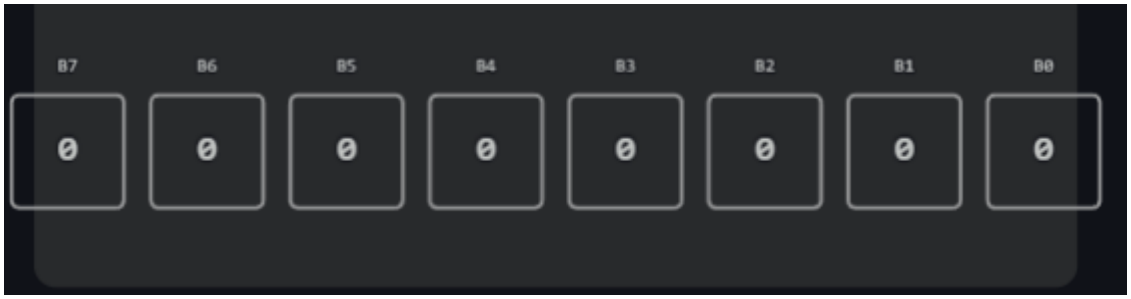


Figura 4. Câmpurile registrului Data_OUT

Registrul DATA_OUT primeste valorile de la Qx de la fiecare client.

Structura registrului de control (**STATUS**) se găsește în Figura 5.

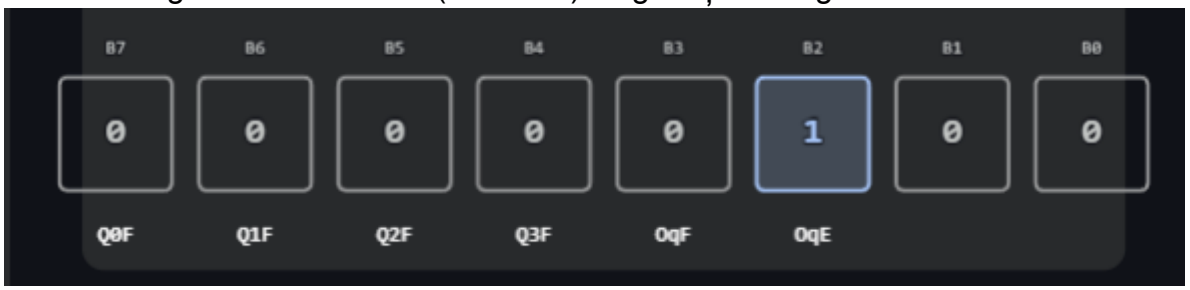


Figura 5. Câmpurile registrului STATUS

- **[7:4] QxF (Queue Full):** Indicatori pentru cozile de intrare (Q0F la bit 7, Q1F la bit 6 etc.).
  - *Constrângere HDL:* Nu se va da ACK către client dacă bitul QxF asociat este 1.
- **[3] OqF (Output Queue Full):** 1 dacă coada de ieșire este plină.
- **[2] OqE (Output Queue Empty):** 1 dacă nu există date pentru citire. (Valoare reset: 1).
- **[1:0] RES:** Rezervați.

Structura registrului de control (CFG) se găsește în Figura 6.

Figura 6. Câmpurile registrului CFG

[5:4] ALG_SEL: Selectarea algoritmului:

- 00: Strict Priority (S.P.)
- 01: Round Robin (R.R.)
- 10: **SLVERR** (O scriere aici va genera o eroare pe magistrala APB).
- 11: Weighted Round Robin (W.R.R.)

[3:0] CxEN: Activare clienți (C3 la bit 3, C0 la bit 0). Reset implicit: 1111 (toți activi).

Structura registrului de control (WEIGHT) se găsește în Figura 7.

Figura 7. Câmpurile registrului WEIGHT

Fiecare câmp de 2 biți reprezintă ponderea (numărul de pachete procesate per tură) pentru clienți.

- **[7:6] W3 | [5:4] W2 | [3:2] W1 | [1:0] W0**
- **TODO: RADU adauga ce inseamna fiecare valoare de la pondere**

1.5 Lista cu funcționalitățile DUT-ului

Acest capitol enumeră detaliat funcționalitățile modulului **Packet Scheduler**, explicând rolul semnalelor de pe interfețe și comportamentul intern al arbitrului. De asemenea, sunt detaliate cazurile limită pentru a ghida inginerii de verificare în realizarea planului de testare și a modelului de referință.

Recepționarea datelor de pe interfața cu clienții (Client 0-3)

Datele sunt primite de la cei 4 clienți printr-un protocol de tip *Request-Acknowledge*. Fiecare client dispune de o interfață dedicată formată din semnalele `req_i`, `data_i` (8 biți) și `ack_o`.

- **Solicitarea transferului:** Un client semnalează intenția de a trimite un pachet activând semnalul `req_i` și menținând datele valide pe magistrala `data_i`.
- **Scrierea în FIFO-ul de intrare:** Modulul dispune de 4 memorii FIFO interne (Queue 0-3), câte una pentru fiecare client. Dacă un client ridică `req_i` și FIFO-ul său asociat *nu este plin*, DUT-ul preia pachetul (`data_i`) și activează semnalul `ack_o` pentru un ciclu de ceas.
- **Gestionarea cozilor pline:** Dacă un client ridică `req_i`, dar FIFO-ul său asociat este plin (indicat intern de semnalul `q_full`), DUT-ul **nu** va acorda semnalul `ack_o`. Pachetul nu este stocat, iar clientul trebuie să mențină cererea activă. Flag-ul corespunzător (ex. `Q0F`) din registrul STATUS va fi setat la 1 pentru a semnala sistemului gazdă congestia.
- **Validarea clientului:** Recepția pachetelor are loc doar dacă clientul respectiv este activat din registrul CFG (biții `C0EN` ... `C3EN` au valoarea 1). Dacă un client este dezactivat, orice cerere (`req_i`) este ignorată complet (nu se acordă `ack_o`).
- **Resetarea modulului:** Activarea semnalului de resetare (`rst_ni` = 0) golește toate cele 4 FIFO-uri de intrare, resetează mașina de stări a arbitrului și aduce regiștrii APB la valorile implicite (ex: toți clienții activați, coada de ieșire marcată ca goală).

Funcționarea blocului de Arbitrare (Scheduler-ul intern)

Sarcina principală a DUT-ului este să selecteze pachetele din cele 4 FIFO-uri de intrare și să le transfere în coada de ieșire (Output FIFO), folosind un algoritm configurabil.

- **Condiții de start:** Arbitrul devine activ dacă există cel puțin un pachet în orice FIFO de intrare valid (client activat) și dacă *Output FIFO nu este plin* (`OqF` = 0).

- **Selectarea canalului (Algoritmul):** Algoritmul de decizie este dictat de biții ALG_SEL din registrul CFG:
    - *Strict Priority (S.P. - 00):* Arbitrul verifică cozile în ordinea strictă: Q0, apoi Q1, apoi Q2, apoi Q3. Dacă Q0 are date, le va procesa continuu până se golește, blocând clienții cu prioritate mai mică.
    - *Round Robin (R.R. - 01):* Arbitrul interoghează cozile secvențial (Q0 -> Q1 -> Q2 -> Q3 -> Q0). Fiecare client are dreptul să transfere un singur pachet pe tură. Dacă o coadă este goală sau clientul este dezactivat, arbitrul sare la următorul.
    - *Weighted Round Robin (W.R.R. - 11):* Arbitrul funcționează ciclic, dar acordă un număr de "credite" fiecărui client, conform valorilor din registrul WEIGHT (W0, W1, W2, W3). De exemplu, dacă W0 = 3, se vor extrage până la 3 pachete din Q0 înainte de a trece la Q1.
- **Transferul intern:** Odată ce un canal este selectat, pachetul este citit din FIFO-ul de intrare și scris în Output FIFO, generând intern semnalele de rd_en pentru coada sursă și wr_en pentru coada destinație.

Transmiterea datelor pe interfața APB (Către Host)

Datele consolidate în Output FIFO sunt citite de către sistemul gazdă prin magistrala standard APB.

- **Mecanismul de citire:** Când sistemul gazdă realizează o tranzacție de citire (Read) la adresa 0x0 (DATA_OUT), DUT-ul extrage pachetul din Output FIFO și îl plasează pe magistrala prdata_o.
- **Gestionarea stării de Gol (Empty):** Dacă Output FIFO este gol (indicat de bitul OqE = 1 din registrul STATUS), citirea de la adresa 0x0 va returna valoarea 0x00.
- **Raportarea Stării:** Sistemul gazdă poate monitoriza starea generală a modulului citind registrul STATUS (0x1), care centralizează flag-urile de Full pentru toate FIFO-urile.

Diagrame de stări (Descriere a funcționării FSM-ului de arbitrare)

Pentru a evidenția logica de extragere a pachetelor, este prezentată mașina de stări a modulului *Scheduler*:

1. **IDLE:** Starea de repaus. FSM-ul așteaptă până când cel puțin o coadă de intrare nu este goală ȘI coada de ieșire nu este plină (OqF == 0).
2. **POLL_QUEUES:** În funcție de ALG_SEL, FSM-ul interoghează starea cozii curente (dictată de pointer-ul Round Robin sau de prioritatea fixă).
3. **CHECK_ENABLE:** Verifică dacă clientul selectat are bitul CxEN activ în registrul CFG. Dacă nu este activ, revine la POLL_QUEUES evaluând următorul client.
4. **TRANSFER_DATA:** Dacă coada selectată are date și clientul este activ, se generează semnalele de citire (rd_en) pentru coada respectivă și scriere (wr_en) pentru Output FIFO. În modul W.R.R., în această stare se decrementează și contorul intern de greutate. FSM-ul se întoarce la POLL_QUEUES sau IDLE în funcție de situație.

Tabele de adevăr

Tabelul de mai jos ilustrează mecanismul de *Handshake* (REQ-ACK) la interfața cu un client generic *x*. Acesta descrie comportamentul semnalelor în funcție de intenția clientului și disponibilitatea spațiului în DUT.

Tabelul 2. Logica de acceptare a pachetelor (Handshake pe canalul X)

req_x_i (Client)	**q_full (Intern DUT)**	**CxEN (APB CFG)**	**ack_x_o**	**Descriere Acțiune**
0	-	-	0	Clientul nu are date de transmis. Stare de repaus.
1	0	1	**1**	Tranzacție de succes. Pachetul este stocat în coada *x*.
1	1	1	**0**	**Limită memorie:** Coada este plină. Pachetul nu este preluat. Clientul trebuie să mențină req_i.
1	-	0	**0**	Clientul este dezactivat din APB. DUT-ul ignoră cererea.

1.6 Forme de undă

În această secțiune sunt prezentate principalele cazuri de funcționare ale modulului **Packet Scheduler**, ilustrând modurile de interacțiune pe magistrala APB (pentru configurare și citire date) și pe interfața dedicată clienților (pentru recepția pachetelor).

Scrierea unui registru de configurare (Interfața APB)

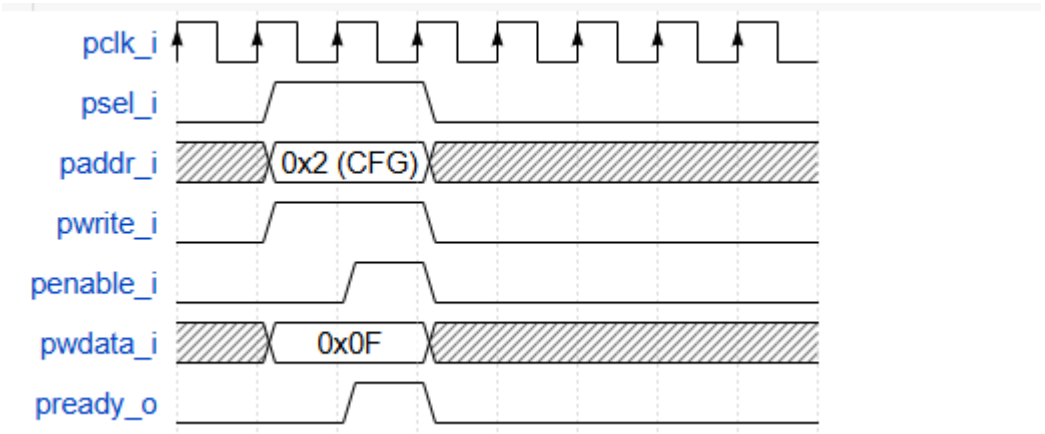
Acest caz ilustrează operația de scriere a magistralei gazdă în registrul de configurare **CFG** (adresa `0x2`), pentru a schimba algoritmul de arbitrare și a activa clienții.

Descrierea operației de scriere (APB Write): Operația de scriere APB necesită două faze (Setup Phase și Access Phase) și implică următoarele semnale:

- `psel_i = 1` – selectează modulul Slave (DUT-ul).
- `pwrite_i = 1` – indică o tranzacție de scriere.
- `paddr_i = 0x2` – indică adresa registrului CFG.
- `penable_i = 1` – validează transferul de date în ciclul al doilea (Access Phase).
- `pwdata_i` – conține valoarea ce trebuie scrisă (ex. `0x0F`).

Desfășurarea operației:

1. Pe frontul crescător al ceasului (`pclk_i`), sistemul gazdă activează `psel_i`, `pwrite_i` și pune adresa `0x2` pe `paddr_i`, intrând în faza de *Setup*.
2. În următorul ciclu de ceas, sistemul gazdă activează `penable_i`.
3. DUT-ul confirmă preluarea datelor activând `pready_o = 1`. Valoarea este memorată în registrul intern.



2. Citirea unui pachet din coada de ieșire (Interfața APB)

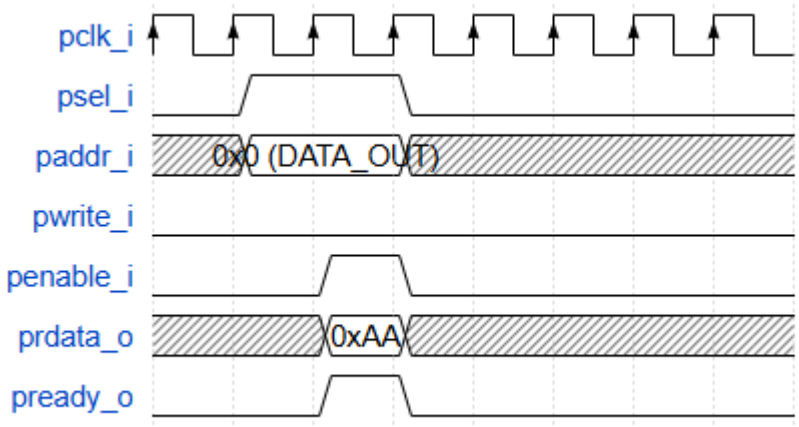
Acest caz ilustrează modul în care sistemul gazdă extrage pachetele arbitrate din **Output FIFO** prin citirea adresei `0x0` (`DATA_OUT`).

Descrierea operației de citire (APB Read):

- `psel_i = 1` și `pwrite_i = 0` – inițiază o tranzacție de citire.
- `paddr_i = 0x0` – adresează portul de citire al cozii centrale.
- `prdata_o` – magistrala pe care DUT-ul va plasa pachetul de date.

Desfășurarea operației:

1. Faza de *Setup* începe prin ridicarea `psel_i` și menținerea `pwrite_i` în `0`, împreună cu setarea adresei la `0x0`.
2. În faza de *Access* (`penable_i` = 1), DUT-ul extrage un element din memorie și îl expune pe `prdata_o`.
3. DUT-ul semnalează validitatea datelor ridicând `pready_o = 1`. Pointer-ul de citire al FIFO-ului intern se incrementează automat.



3. Recepționarea datelor de la client (Protocol REQ/ACK)

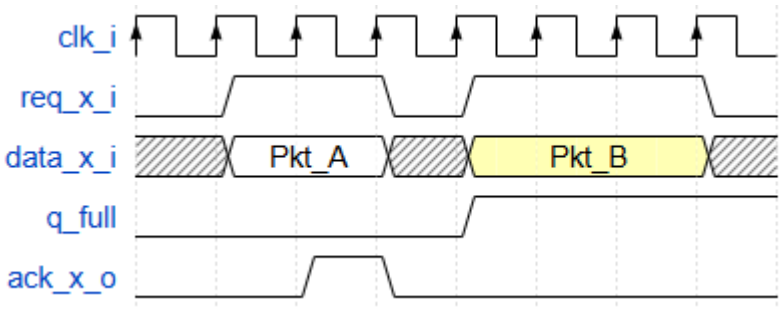
Această diagramă evidențiază logica de acceptare a pachetelor de la un client oarecare. Sunt ilustrate două scenarii: o tranzacție de succes și o tranzacție respinsă din cauza cozii pline.

Descrierea operației:

- `req_x_i = 1` – Clientul are date valide și dorește să le transfere.
- `data_x_i` – Conține pachetul de 8 biți.
- `q_full` – Semnal intern al DUT-ului; indică faptul că FIFO-ul de intrare pentru acest client nu mai are spațiu.
- `ack_x_o` – Răspunsul DUT-ului.

Desfășurarea operației:

1. **Tranzacția cu succes:** Clientul activează `req_x_i`. Deoarece coada internă are spațiu (`q_full = 0`), DUT-ul înregistrează datele și activează `ack_x_o` în ciclul imediat următor.
2. **Tranzacția blocată (Limită memorie):** Mai târziu, memoria FIFO internă se umple (`q_full = 1`). Când clientul inițiază o nouă cerere (`req_x_i = 1`), DUT-ul observă lipsa spațiului și **nu** generează `ack_x_o`. Clientul este obligat să țină cererea activă (wait state) până la eliberarea memoriei.



de modificat, pentru a se indeplini regulile la protocolului request - acknowledge TODO:RADU

4. Eroare la scrierea unui algoritm rezervat (APB Slave Error)

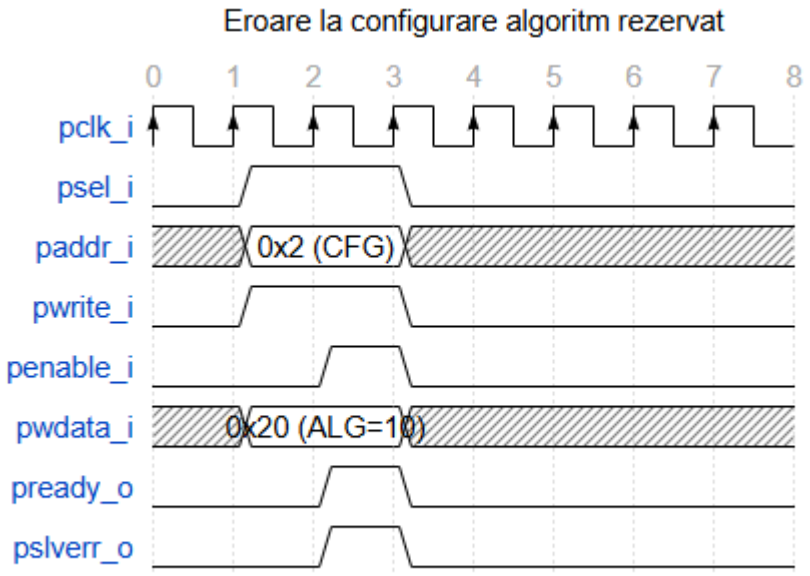
Acest caz ilustrează situația în care sistemul gazdă încearcă să configureze registrul **CFG** (adresa `0x2`) cu o valoare nepermisă pentru câmpul `ALG_SEL` (valoarea `2'b10`).

Descrierea operației de eroare (APB SLVERR):

- `paddr_i = 0x2` – indică adresa registrului CFG.
- `pwdata_i = 8'b00100000` – bitul 5:4 au valoarea `10` (valoare rezervată conform specificației).
- `pslverr_o = 1` – semnalul de eroare generat de DUT pentru a indica o tranzacție invalidă.

Desfășurarea operației:

1. Master-ul inițiază faza de *Setup* cu adresa `0x2` și datele invalide.
2. La faza de *Access* (`penable_i = 1`), logica de decodificare internă a DUT-ului detectează faptul că valoarea scrisă pentru algoritm este interzisă.
3. DUT-ul activează simultan `pready_o = 1` și `pslverr_o = 1` pentru a semnala eșecul scrierii.
4. Registrul intern **CFG** își păstrează valoarea anterioară, ignorând noua scriere.



2 VERIFICAREA PROIECTULUI DIGITAL AL CIRCUITULUI

Arhitectura mediului de verificare
Lista cu elementele de verificat
Elementele de acoperire funcțională
Scenarii de verificare și teste
Rezultate obținute

2.1 ARHITECTURA MEDIULUI DE VERIFICARE

Se realizează o imagine de ansamblu cu mediul de verificare.

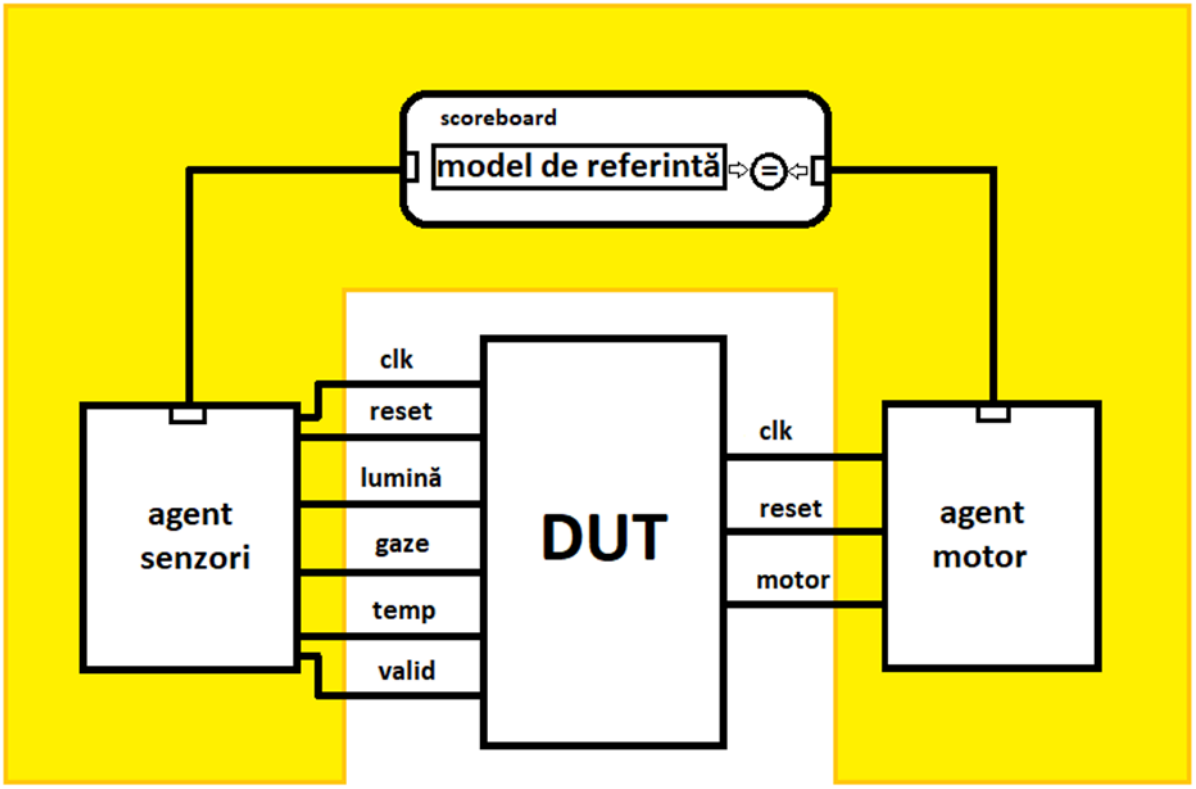


Figura 3. Exemplu de imagine de ansamblu a mediului de verificare

Se realizează câte un subcapitol pentru fiecare componentă

- se descrie cum funcționează fiecare componentă
- se adaugă o schemă logică pentru componente

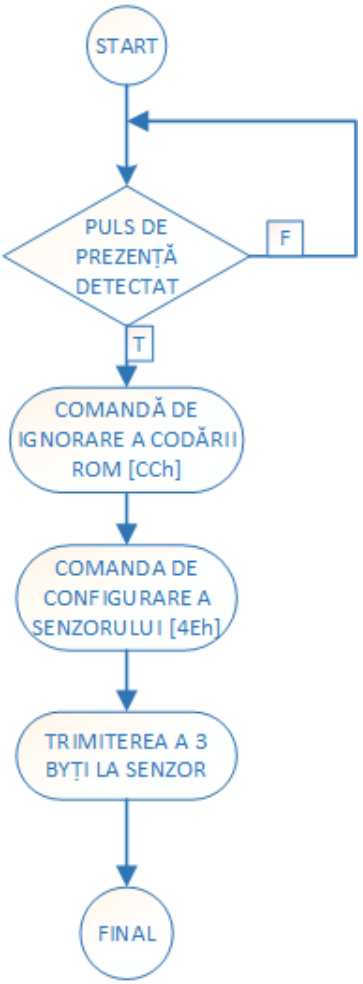


Figura 5. Exemplu de schemă logică

Exemplu de titluri de subcapitole corespunzătoare figurilor 3 si 4.

2.2 Lista cu elementele de verificat

Situații de funcționare ale modulului (Cazuri limită pentru Verificare)

Pentru a asigura validarea exhaustivă a DUT-ului, planul de verificare trebuie să acopere următoarele scenarii:

- **Suprasaturarea intrărilor (Queue Full pe toate canalele):** Se generează cereri continue pe toți cei 4 clienți într-un ritm mai rapid decât poate procesa arbitrul sau poate citi APB-ul. Se verifică faptul că semnalele `ack_o` nu mai sunt acordate, iar flag-urile `Q0F`-`Q3F` din registrul STATUS sunt ridicate corect.
- **Înfometarea (Starvation) în modul S.P.:** Se configurează algoritmul pe S.P. și se trimite trafic continuu pe Clientul 0. Se verifică faptul că pachetele de pe Clienții 1, 2 și 3 nu sunt procesate (rămân blocate în cozile lor).
- **Schimbarea algoritmului *On-the-fly*:** Se modifică biții `ALG_SEL` din APB în timp ce există pachete în cozile de intrare. Se verifică dacă mașina de stări a arbitrului face tranziția corectă la noua logică de planificare fără a pierde sau duplica pachete.
- **Ponderi WRR epuizate prematur:** În modul W.R.R., se setează `W0` = 3, dar se introduc doar 2 pachete în Q0. Se verifică dacă arbitrul nu rămâne blocat așteptând al treilea pachet, ci trece corect la Q1 după golirea lui Q0.
- **Citire și Scriere simultană în Output FIFO:** Arbitrul decide să transfere un pachet din Q_in -> Output FIFO exact în același ciclu de ceas în care interfața APB realizează o citire de la `0x0`. Se verifică actualizarea corectă a pointerilor FIFO-ului de ieșire (numărul de elemente rămâne constant).
- **Eroare de adresare APB:** Se încearcă scrierea la adresa `0x2` (CFG) cu valoarea `ALG_SEL` = `10` (mod rezervat). Se verifică generarea semnalului `pslverr_o` către magistrala APB.

Se descrie cum se verifică faptul că DUT-ul funcționează bine.
Elementele de verificat vor fi organizate in functie de componentele in care sunt scrise (interfețe / monitoare/ scoreboard). Se va realiza cate un tabel pentru fiecare componenta.

Tabelul 5. Checkerele de pe interfata de iesire

Checker	Descriere
burst_len_check	Verifica daca numarul de cicli primiti este conform cu burst length primit pe interfata de request
data_integrity_check	Verifica daca datele primite la iesire sunt conform cu cele asteptate (de ex. datele de la intrare impachetate intr-un anumit fel).
burst_addr_check	Verifica daca adresa pentru burst-ul curent este adresa burst-ului precedent + numarul de Bytes transmisi pentru acel burst.

2.3 Elementele de acoperire funcțională

Elementele de *coverage* se pot enumera sub formă de listă sau tabel (fiecare grup de coverage intr-un tabel separat). Acestea vor fi, de asemenea, organizate in funcție de componente/interfețe.

Tabelul 6. Elementele de acoperire funcțională pentru datele de pe interfața de ieșire

Eveniment: cover_trans_done_ev: emis la finalul unei tranzactii burst	
Cover Point	**Descriere**
burst_len_cp	Burst length a luat toate valorile de la 1 la 16
burst_addr_cp	Tot range-ul de adrese a fost acoperit. Adresele sunt pe 32 de biti, se considera adresa minima, maxima, iar restul range-ului este impartit in 32 de parti egale
burst_rd_wr_cp	S-au facut si burst-uri de read si burst-uri de write.
cross_burst_len_rd_wr_cp	Cross intre burst_len_cp si read/write.

2.4 Scenarii de verificare și teste

Se va realiza un tabel cu scenariile de verificare propuse și cu testele implementate

Tabelul 7. Lista scenariilor proiectate pentru verificarea DUT-ului

Titlul scenariului	Descrierea scenariului propus
resetarea registrilor	Se vor citi valorile registrilor imediat dupa ce semnalul reset a fost

	activ, pentru a se verifica valoarea acestora
scrierea registrilor	Se scriu toti registrii cu valori aleatorii, si apoi se citescf
activarea si dezactivarea alarmei de incendiu	1. Se seteaza in registru o valoare de prag pentru temperatura. 2. Se trimit pe interfata de intrare valori de temperatura care sa depaseasca valoarea de prag setata in registru. 3. Dupa ce se observa ca alarma s-a activat se trimit pe intrare alte valori de temperatura sub valoarea de prag. 4. Se sterge bitul de alarma din registru.
umplerea cu date a unui fifo pentru un client si dezactivarea acestuia	……

Tabelul 8. Lista testelor implementate pentru verificarea DUT-ului

Numele fișierului care conține testul	**Descrierea testului realizat**
test_overflow.sv	În cadrul acestui test se dau valori mari părților reale și imaginare ale celor doi operanzi astfel încât rezultatele obținute să nu poată fi reprezentate doar pe 8 biți.
test_latency.sv	În cadrul acestui test se schimbă numărul de tacte necesar pentru a se efectua un calcul de mai multe ori, în timpul desfășurării operațiilor.
……	……

2.5 Rezultate obținute

Rezultatele privesc acoperirea funcțională desfășurarea proiectelor per ansamblu și alte aspecte de interes.

În urma rulării testului test_complete.sv s-a obținut valoarea de 100% pentru elementul de acoperire funcțională privind valorile latenței.